

# Two RPG Flow-graphs for Software Watermarking using Bitonic Sequences of Self-inverting Permutations

Anna Mpanti   Stavros D. Nikolopoulos

*Department of Computer Science & Engineering*  
*University of Ioannina*  
*GR-45110 Ioannina, Greece*  
 {ampanti,stavros}@cs.uoi.gr

## Abstract

Software watermarking has received considerable attention and was adopted by the software development community as a technique to prevent or discourage software piracy and copyright infringement. A wide range of software watermarking techniques has been proposed among which the graph-based methods that encode watermarks as graph structures. Following up on our recently proposed methods for encoding watermark numbers  $w$  as reducible permutation flow-graphs  $F[\pi^*]$  through the use of self-inverting permutations  $\pi^*$ , in this paper, we extend the types of flow-graphs available for software watermarking by proposing two different reducible permutation flow-graphs  $F_1[\pi^*]$  and  $F_2[\pi^*]$  incorporating important properties which are derived from the bitonic subsequences composing the self-inverting permutation  $\pi^*$ . We show that a self-inverting permutation  $\pi^*$  can be efficiently encoded into either  $F_1[\pi^*]$  or  $F_2[\pi^*]$  and also efficiently decoded from these graph structures. The proposed flow-graphs  $F_1[\pi^*]$  and  $F_2[\pi^*]$  enrich the repository of graphs which can encode the same watermark number  $w$  and, thus, enable us to embed multiple copies of the same watermark  $w$  into an application program  $P$ . Moreover, the enrichment of that repository with new flow-graphs increases our ability to select a graph structure more similar to the structure of a given application program  $P$  thereby enhancing the resilience of our codec system to attacks.

**Keywords:** Watermarking, self-inverting permutation, reducible permutation graphs, codec algorithms, encoding, decoding.

## 1 Introduction

Over the last 25 years, digital or multimedia watermarking has become a popular technique for protecting the intellectual property of any digital content such as image, audio, video or software data. In this domain, software watermarking has received considerable attention and was adopted by the software development community as a technique to prevent or discourage software piracy and copyright infringement. A wide range of software watermarking techniques has been proposed among which the graph-based methods that encode watermark numbers as graphs whose structure resembles that of real program graphs.

**Watermarking.** Digital watermarking is a popular technique for copyright protection of a digital object or, in general, multimedia information [17, 19]; the idea is simple: a unique marker, which is called watermark, is embedded into a digital object which may be used to verify its authenticity or the identity of its owners [5, 11]. The software watermarking problem can be described as the

problem of embedding a structure  $w$  into a program  $P$  such that  $w$  can be reliably located and extracted from  $P$  even after  $P$  has been subjected to code transformations such as translation, optimization and obfuscation [3, 15]. More precisely, given a program  $P$ , a watermark  $w$ , and a key  $k$ , the software watermarking problem can be formally described by the following two functions:  $embed(P, w, k) \rightarrow P$  and  $extract(P_w, k) \rightarrow w$ .

Since the late 1990s, there has been an explosion in the number of digital watermarking techniques among which time-series, biological sequences, graph-structured data, spatial data, spatiotemporal data, data-streams and others [16]. Recently, software watermarking has received considerable attention and many researchers have developed several codec algorithms mostly for watermarks that are encoded as graph-structures [9]. The patent by Davidson and Myhrvold [8] presented the first published software watermarking algorithm. The preliminary concepts of software watermarking also appeared in paper [10] and patents [14, 17]. Collberg et al. [6, 7] presented detailed definitions for software watermarking. Authors of papers [20, 21] have given brief surveys of software watermarking research.

**Our Contribution.** Recently, we have presented codec algorithms, namely `Encode_W.to.SIP` and `Decode_SIP.to.W`, for encoding an integer  $w$  into a self-inverting permutation  $\pi^*$  and extracting it from  $\pi^*$  [1], and several codec algorithms for encoding  $\pi^*$  into many reducible permutation flow-graphs  $F_i[\pi^*]$  ( $i > 1$ ) [2, 4], having thus created a wide repository of graph-structures, namely flow-graphs, whose structures resemble that of real program graphs.

In this paper, we extend the types of flow-graphs which can efficiently encode a self-inverting permutation  $\pi^*$  by proposing two different reducible permutation flow-graphs  $F_1[\pi^*]$  and  $F_2[\pi^*]$  having properties which are derived from the bitonic subsequences  $b_1^*, b_2^*, \dots, b_k^*$  composing the self-inverting permutation  $\pi^*$ . We show relations between the elements of such a bitonic subsequence  $b_i^*$  and their indices in  $\pi^*$  and prove properties for the first, last, max and min elements of  $\pi^*$ . We also show that the first bitonic subsequence  $b_1^*$  of a self-inverting permutation  $\pi^*$  of length  $n^*$  has always length  $\lceil n^*/2 \rceil$  and structure  $(\lceil n^*/2 \rceil, \dots, \pi_{max}^*, \dots, 1)$ , where  $\pi_{max}^*$  is the max element of  $\pi^*$ .

Taking advantage of these properties, we construct two different reducible permutation flow-graphs  $F_1[\pi^*]$  and  $F_2[\pi^*]$  which can encode the same self-inverting permutation  $\pi^*$  and thus, the same watermark number  $w$ . By construction, the indegree of the first node  $s = u_{n^*+1}$  of the flow-graph  $F_1[\pi^*]$  is equal to the number of bitonic subsequences  $b_1^*, b_2^*, \dots, b_k^*$  of  $\pi^*$ , while the indegree of the first node of the graph  $F_2[\pi^*]$  is much smaller than  $k$ . This property causes  $F_2[\pi^*]$  more appropriate, in some cases, since it does not contain an extreme characteristic thereby enhancing the resilience of graph-structure to attacks.

The flow-graphs  $F_1[\pi^*]$  and  $F_2[\pi^*]$  enrich the repository of graphs which can encode the same watermark number  $w$  and, thus, enable us to embed several copies of the same watermark  $w$  into an application program  $P$ . Moreover, it increases our ability to select a graph structure more similar to the structure of a given application program  $P$  thereby enhancing the resilience of our codec system to attacks.

**Road Map.** The paper is organized as follows: In Section 2 we establish the notation and related terminology, we present background results, and show properties of the bitonic subsequences which compose a self-inverting permutation  $\pi^*$ . In Sections 3 and 4 we present our two codec algorithms for encoding a self-inverting permutation  $\pi^*$  into two different reducible permutation flow-graphs  $F_1[\pi^*]$  and  $F_2[\pi^*]$  having properties deriving from the bitonic subsequences composing the self-inverting permutation  $\pi^*$ . We show that the permutation  $\pi^*$  can be efficiently encoded into either  $F_1[\pi^*]$  or  $F_2[\pi^*]$  and also correctly and efficiently extracted from these flow-graphs. Finally, in Section 5 we conclude the paper and discuss possible future extensions.

## 2 Theoretical Framework

We consider finite graphs with no multiple edges. For a graph  $G$ , we denote by  $V(G)$  and  $E(G)$  the vertex (or, node) set and edge set of  $G$ , respectively. The subgraph of a graph  $G$  induced by a set  $S \subseteq V(G)$  is denoted by  $G[S]$ . The *neighborhood*  $N(x)$  of a vertex  $u$  of the graph  $G$  is the set of all the vertices of  $G$  which are adjacent to  $u$ . The *degree* of a vertex  $u$  in the graph  $G$ , denoted  $\deg(u)$ , is the number of edges incident on node  $u$ ; for a node  $u$  of a directed graph  $G$ , the number of head-endpoints of the directed edges adjacent to  $u$  is called the indegree of the node  $u$ , denoted  $\text{indeg}(u)$ , and the number of tail-endpoints is its outdegree, denoted  $\text{outdeg}(u)$ . The parent of a node  $x$  of a rooted tree  $T$  is denoted by  $p(x)$ .

### 2.1 Previous Results

In mathematics, the notion of permutation relates to the act of arranging all the members of a set into a sequence or order. Permutations may be represented in many ways [18], where the most straightforward is simply a rearrangement of the elements of the set  $N_n = \{1, 2, \dots, n\}$ . For example,  $\pi = (5, 6, 8, 9, 1, 2, 7, 3, 4)$  is a permutation of the elements of the set  $N_9$ ; hereafter, we shall say that  $\pi$  is a permutation over the set  $N_9$ .

**Definition 2.1** Let  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$  be a permutation over the set  $N_n$ ,  $n > 1$ . The inverse of the permutation  $\pi$  is the permutation  $q = (q_1, q_2, \dots, q_n)$  with  $q_{\pi_i} = \pi_{q_i} = i$ . A *self-inverting permutation* (or, for short, SiP) is a permutation that is its own inverse:  $\pi_{\pi_i} = i$ .

By definition, a permutation is a SiP (self-inverting permutation) if and only if all its cycles are of length 1 or 2; for example, the permutation  $\pi = (5, 6, 8, 9, 1, 2, 7, 3, 4)$  is a SiP with cycles:  $(1, 5)$ ,  $(2, 6)$ ,  $(3, 8)$ ,  $(4, 9)$ , and  $(7)$ . Throughout the paper we shall denote a self-inverting permutation  $\pi$  over the set  $N_n$  as  $\pi^*$ .

A flow-graph is a directed graph  $F$  with an initial node  $s$  from which all other nodes are reachable. A directed graph  $G$  is strongly connected when there is a path  $x \rightarrow y$  for all nodes  $x, y$  in  $V(G)$ . A node  $u \in V(G)$  is an *entry* for a subgraph  $H$  of the graph  $G$  when there is a path  $p = (y_1, y_2, \dots, y_k, x)$  such that  $p \cap H = \{x\}$  (see, [12, 13]).

**Definition 2.2** A flow-graph is reducible when it does not have a strongly connected subgraph with two (or more) entries.

There are some other equivalent definitions of the reducible flow-graphs which use a few more graph-theoretic concepts. A depth first search (DFS) of a flow-graph partitions its edges into tree, forward, back, and cross edges. It is well known that tree, forward, and cross edges form a dag known as a DFS dag. Hecht and Ullman show that a flow-graph  $F$  is reducible if and only if  $F$  has a unique DFS dag [12, 13].

Recently, a wide range of software watermarking techniques has been proposed among which the graph-based methods that encode watermark numbers  $w$  as reducible flow-graph structures  $F$  capturing such properties which make them resilient to attacks.

In [1], Chroni and Nikolopoulos presented the encoding algorithm `Encode_W.to_SIP`, along with its corresponding decoding one, which encodes a watermark number  $w$  as a self-inverting permutation  $\pi^*$  in  $O(n)$  time, where  $n$  is the length of the binary representation of the integer  $w$ . Later, the same authors introduced a type of reducible permutation graphs  $F[\pi^*]$  and proposed several efficient codec methods which embed a self-inverting permutation  $\pi^*$  into such a reducible permutation graph  $F[\pi^*]$  and efficiently extract it from the graph  $F[\pi^*]$ . Moreover, they proposed several algorithms for multiple encoding the same watermark number  $w$  into many different reducible permutation graphs  $F_i[\pi^*]$ ,  $i > 1$ , through the use of the encoding self-inverting permutation  $\pi^*$  [2, 4].

These results are summarized in the following theorems and lemmata.

**Theorem 2.1** *Let  $w$  be an integer and let  $b_1b_2\cdots b_n$  be the binary representation of  $w$ . The number  $w$  can be encoded into a self-inverting permutation  $\pi^*$  of length  $2n+1$  and correctly extracted from  $\pi^*$  in  $O(n)$  time and space.*

**Theorem 2.2** *Let  $\pi^*$  be a self-inverting permutation of length  $n^*$  which encodes a watermark integer  $w$ . The permutation  $\pi^*$  can be encoded as a reducible permutation flow-graph  $F[\pi^*]$  and correctly extracted from  $F[\pi^*]$  in  $O(n^*)$  time and space.*

**Lemma 2.1** *Let  $F[\pi^*]$  be a reducible permutation graph of size  $O(n^*)$  constructed by a proper encoding algorithm. The unique Hamiltonian path of  $F[\pi^*]$  can be computed in  $O(n^*)$  time and space.*

**Theorem 2.3** *We can produce more than one reducible flow-graphs  $F_1[\pi^*], F_2[\pi^*], \dots, F_n[\pi^*]$  which encode the same watermark integer  $w$  through the use of the self-inverting permutation  $\pi^*$ .*

## 2.2 Bitonic Sequences and SiPs

A sequence  $b = (b_1, b_2, \dots, b_n)$  is called bitonic if either monotonically increases and then monotonically decreases, or else monotonically decreases and then monotonically increases.

In this paper, we consider only bitonic sequences that monotonically increases and then monotonically decreases, i.e., the minimum element of such a sequence  $b$  is either the first  $b_1$  or the last  $b_n$  element of  $b$ ; for example,  $b = (5, 6, 8, 9, 1)$  is such a bitonic sequence. The maximum element of a bitonic sequence  $b$ , which we call *top* element of  $b$ , is denoted as  $top(b)$ . Obviously,  $b$  is an increasing sequence if  $top(b) = b_n$ , while  $b$  is a decreasing sequence if  $top(b) = b_1$ .

**Definition 2.3** Let  $b = (b_1, b_2, \dots, b_n)$  be a bitonic sequence of length  $n$ . According to the index of the element  $top(b)$ , the sequence  $b$  is called:

- *i*-bitonic or *increasing bitonic* if  $top(b) = b_n$ ;
- *d*-bitonic or *decreasing bitonic* if  $top(b) = b_1$ ;
- *id*-bitonic or *full-bitonic* if  $b_1 < top(b)$  and  $top(b) > b_n$ .

In terms of the above definition,  $b_1 = (2, 7)$  is an *i*-bitonic or increasing bitonic sequence,  $b_2 = (4, 3)$  is a *d*-bitonic or decreasing bitonic sequence, while  $b_3 = (5, 6, 8, 9, 1)$  is an *id*-bitonic or full-bitonic sequence.

Let  $\pi^*$  be the self-inverting permutation of length  $n^*$  produced by the encoding algorithm **Encode\_W.to.SiP** [1], and let  $b_1^*, b_2^*, \dots, b_k^*$  be the bitonic subsequences forming the permutation  $\pi^*$ ; note that,  $\pi^*$  encodes a watermark number  $w$  of binary length  $n$  and  $n^* = 2n + 1$ . Then,  $b_1^*, b_2^*, \dots, b_k^*$  have the following properties:

- ( $P_1$ ) Sequence  $b_1^*$  is a full-bitonic,  $b_2^*, b_3^*, \dots, b_{k-1}^*$  are either full-bitonic or *d*-bitonic sequences, while  $b_k^*$  is either a full-bitonic, *i*-bitonic, or *d*-bitonic sequence.
- ( $P_2$ ) Sequence  $b_1^*$  contains the max element  $\pi_{max}^*$  and the min element  $\pi_{min}^*$  of permutation  $\pi^*$  and has always length  $\lceil n^*/2 \rceil$ ; note that,  $\pi_{max}^* = top(b_1^*) = 2n + 1$  and  $\pi_{min}^* = 1$ ;
- ( $P_3$ ) The last element  $last(b_k^*)$  of sequence  $b_k^*$  is equal to the index of the max element  $\pi_{max}^* = 2n + 1$  in  $b_1^*$ .

**Example 2.1** Let  $w_1 = 20$  and  $w_2 = 45$  be two watermark numbers. For these two watermarks, the encoding algorithm `Encode_W.to.SIP` produces the self-inverting permutations

- $\pi_1^* = (6, 8, 11, 10, 9, 1, 7, 2, 5, 4, 3)$ , and
- $\pi_2^* = (7, 9, 10, 12, 13, 11, 1, 8, 2, 3, 6, 4, 5)$

of lengths  $n_1^* = 2n_1 + 1 = 11$  and  $n_2^* = 2n_2 + 1 = 13$ , respectively; note that,  $n_1 = 5$  is the length of the binary representation of number 20 (i.e., 10100), while  $n_2 = 6$  is that of number 44 (i.e., 101101). The permutations  $\pi_1^*$  and  $\pi_2^*$  are composed by the following three and four, respectively, bitonic subsequences:

- $\pi_1^* : (6, 8, 11, 10, 9, 1) \parallel (7, 2) \parallel (5, 4, 3)$
- $\pi_2^* : (7, 9, 10, 12, 13, 11, 1) \parallel (8, 2) \parallel (3, 6, 4) \parallel (5)$

We observe that all the bitonic subsequences of both permutations  $\pi_1^*$  and  $\pi_2^*$  satisfy the properties  $(P_1)$ ,  $(P_2)$ , and  $(P_3)$ . Indeed, for example, the subsequence  $b_1^* = (6, 8, 11, 10, 9, 1)$  of permutation  $\pi_1^*$  is full-bitonic, contains the max and the min elements of  $\pi_1^*$ , has length  $6 = \lceil n_1^*/2 \rceil$ , where  $n_1^* = 11$  is the length of  $\pi_1^*$ , and the last element of subsequence  $b_3^* = (5, 4, 3)$  (i.e., 3) is equal to the index (i.e., 3) of the max element of  $\pi_1^*$  in sequence  $b_1^*$  (i.e.,  $\pi_{max}^* = 11$ ).

**Example 2.2** Here is an example of the permutation  $\pi^*$  which encodes the number  $w = 54$  with binary representation 110110, i.e.,  $\pi^* = (7, 8, 10, 11, 13, 12, 1, 2, 9, 3, 4, 6, 5)$ . It is easy to see that this permutation satisfies the properties  $(P_1) - (P_3)$  and all its bitonic subsequences are of type full-bitonic; indeed,  $b_3^* = (7, 8, 10, 11, 13, 12, 1)$ ,  $b_2^* = (2, 9, 3)$ , and  $b_1^* = (4, 6, 5)$ .

### 3 Bitonic Algorithm

Having presented an efficient codec algorithm for encoding a watermark number  $w$  as a self-inverting permutation  $\pi^*$  [1] and several codec algorithms for efficiently encoding the permutation  $\pi^*$  into different reducible permutation flow-graphs  $F_i[\pi^*]$  ( $i > 1$ ), in this section we extend the types of such flow-graphs by proposing an algorithm for encoding a self-inverting permutation  $\pi^*$  into a reducible permutation graph  $F[\pi^*]$  having properties which are derived from the bitonic subsequences composing the self-inverting permutation  $\pi^*$  (see, properties  $P_1 - P_3$ ).

The encoding algorithm, which we call `Encode_SIP.to.RPG-Bitonic-1` is described below.

**Algorithm** `Encode_SIP.to.RPG-Bitonic-1`

1. Compute the bitonic subsequences  $S_1, S_2, \dots, S_k$  of the self-inverting permutation  $\pi^*$  and let  $S_i = (i_1, i_2, \dots, top(S_i), \dots, i_t)$ ;
2. Construct a directed graph  $F_1[\pi^*]$  on  $n^* + 2$  vertices as follows:
  - 2.1  $V(F_1[\pi^*]) = \{s = u_{n^*+1}, u_{n^*}, \dots, u_1, u_0 = t\}$ ;
  - 2.2 for  $i = n^*, n^* - 1, \dots, 0$  do  
add the edge  $(u_{i+1}, u_i)$  in  $E(F_1[\pi^*])$ ;
3. For each bitonic subsequence  $S_i$ ,  $1 \leq i \leq k$ , do
  - 3.1 add the edge  $(u_{top(S_i)}, s)$  in  $E(F_1[\pi^*])$ ;
  - 3.2 for  $j = 1, 2, \dots, top(S_i), \dots, t - 1$  do

if  $i_j < i_{j+1}$  then add the edge  $(u_{i_j}, u_{i_{j+1}})$   
else the edge  $(u_{i_{j+1}}, u_{i_j})$  in  $E(F_1[\pi^*])$ ;

4. Return the graph  $F_1[\pi^*]$ ;

Figure 1 shows the encoding of the self-inverting permutation  $\pi^* = (6, 8, 11, 10, 9, 1, 7, 2, 5, 4, 3)$  into the reducible permutation flow-graph  $F_1[\pi^*]$ ; note that,  $\pi^*$  encodes the watermark number  $w = 20$ .

Let  $w$  be a watermark number and  $\pi^*$  be the self-inverting permutation of length  $n^* = 2n + 1$  which encodes watermark  $w$ , where  $n$  is the length of the binary representation of number  $w$ . Analyzing the time and space performance of our encoding algorithm `Encode_SIP.to.RPG-Bitonic-1`, we can obtain the following result:

**Theorem 3.1** *The algorithm `Encode_SIP.to.RPG-Bitonic-1` for encoding the permutation  $\pi^*$  as a reducible permutation flow-graph  $F_1[\pi^*]$  requires  $O(n)$  time and space.*

We next present the decoding algorithm `Decode_RPG.to.SIP-Bitonic-1`, which takes as input a reducible permutation flow-graph  $F_1[\pi^*]$  on  $n^* + 2$  nodes produced by algorithm `Encode_SIP.to.RPG-Bitonic-1` and correctly extract the self-inverting permutation  $\pi^*$  from the graph  $F_1[\pi^*]$ ; the algorithm is described below.

**Algorithm `Decode_RPG.to.SIP-Bitonic-1`**

1. Delete the directed edges  $(u_{i+1}, u_i)$ ,  $1 \leq i \leq n$ , and the node  $t = u_0$  from  $F_1[\pi^*]$ , and flip all the remaining directed edges in  $F_1[\pi^*]$ ; let  $s = u_0, u_1, u_2, \dots, u_n$  be the nodes in the resulting graph  $T_1[\pi^*]$ ;
2. Compute the set  $R = \{u_j \mid (s, u_j) \in T_1[\pi^*]\}$  and delete the directed edges  $(s, u_j)$  from the graph  $T_1[\pi^*]$ ;
3. Sort the nodes of set  $R$  in descending order according to their labels and let  $R^* = (r_1, r_2, \dots, r_k)$  be the resulting sorted sequence,  $1 \leq k < n$ ;
4. Construct the underlying graph  $H[\pi^*]$  of the directed graph  $T_1[\pi^*]$  and let  $C(r_1), C(r_2), \dots, C(r_k)$  be the connected components of the graph  $H[\pi^*]$  which contain the nodes  $r_1, r_2, \dots, r_k$ , respectively;
5. For each node  $r_i \in R^*$ ,  $i = 1, 2, \dots, k$ , perform BFS-search in graph  $C(r_i)$  starting at node  $u$  and compute the sequence  $b_i^*$  of the nodes of  $C(r_i)$  taken by the order in which they are BFS-discovered; the starting node  $u$  is selected as follows:
  - if  $i < k$  and  $\deg(r_i) = 2$ , then  $u$  is the node with minimum label in  $C(r_i)$ ;
  - if  $i < k$  and  $\deg(r_i) \leq 1$ , then  $u$  is the node with maximum label in  $C(r_i)$ ;
  - if  $i = k$ , then  $u$  is the node with label  $\ell_{max}$  in  $C(r_i)$ , where  $\ell_{max}$  is the index of the max element in sequence  $(b_1^*)^R$ ;

recall that,  $(b_i^*)^R$  denotes the reverse sequence of  $b_i^*$ ,  $1 \leq i \leq k$ ;

6. Return  $\pi^* = (b_1^*)^R || b_2^* || \dots || b_{k-1}^* || (b_k^*)^R$ ;

Figure 1 shows the extraction of the self-inverting permutation  $\pi^* = (6, 8, 11, 10, 9, 1, 7, 2, 5, 4, 3)$ , which encodes the watermark number  $w = 20$ , from the reducible permutation flow-graph  $F_1[\pi^*]$  using the tree  $T_1[\pi^*]$ . Note that, a reducible permutation flow-graph  $F_1[\pi^*]$  of size  $n^* + 2$  encodes a self-inverting permutation  $\pi^*$  of length  $n^*$ .

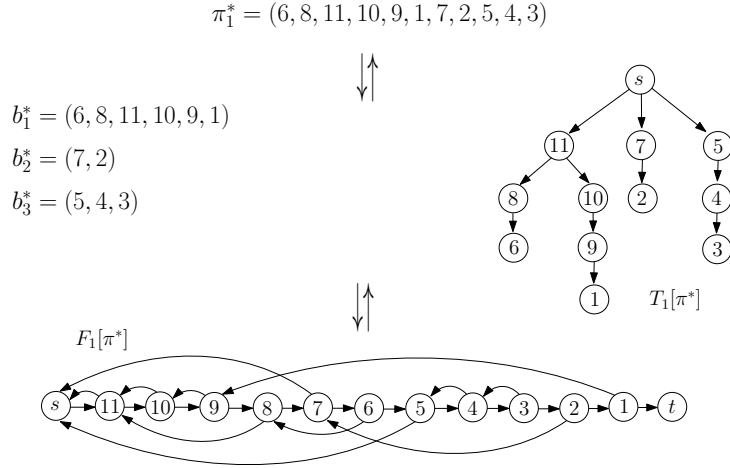


Figure 1: The main structures used or constructed by the codec algorithms `Encode_SiP.to.RPG-Bitonic-1` and `Decode_RPG.to.SiP-Bitonic-1`.

The following result summarizes the correctness and the time and space complexity of our proposed decoding algorithm `Decode_RPG.to.SiP-Bitonic-1`.

**Theorem 3.2** *The algorithm `Decode_RPG.to.SiP-Bitonic-1` correctly extract the permutation  $\pi^*$  from a reducible permutation flow-graph  $F_1[\pi^*]$  in  $O(n)$  time and space.*

## 4 Full-Bitonic Algorithm

In this section we enrich the repository of reducible permutation flow-graphs  $F[\pi^*]$  which can encode a self-inverting permutation  $\pi^*$  or, equivalently, a watermark number  $w$  by proposing a reducible permutation flow-graph  $F_2[\pi^*]$ , different from  $F_1[\pi^*]$  but of the same type, having also important properties deriving from the bitonic subsequences of  $\pi^*$ .

By construction, the indegree of the first node  $s = u_{n^*+1}$  of the flow-graph  $F_1[\pi^*]$  is equal to the number of bitonic subsequences  $b_1^*, b_2^*, \dots, b_k^*$  of  $\pi^*$ , while the indegree of the first node of the graph  $F_2[\pi^*]$  is much smaller than  $k$ . This property causes  $F_2[\pi^*]$  more appropriate, in some cases, since it does not contain an extreme characteristic thereby enhancing the resilience of graph-structure to attacks. The proposed algorithm `Encode_SiP.to.RPG-Bitonic-2` for encoding a self-inverting permutation  $\pi^*$  into a reducible permutation graph  $F_2[\pi^*]$  is described below.

### Algorithm `Encode_SiP.to.RPG-Bitonic-2`

1. Execute algorithm `Encode_SiP.to.RPG-Bitonic-1` and compute the bitonic subsequences  $S_1, S_2, \dots, S_k$  of  $\pi^*$  and the graph  $F_1[\pi^*]$ ; Set  $F_2[\pi^*] \leftarrow F_1[\pi^*]$ ;
2. For each edge  $(u_{top(S_i)}, s)$  in  $F_2[\pi^*]$ ,  $2 \leq i \leq k$ , do  
if  $S_i$  is a full-bitonic sequence, then
  - delete the edge  $(u_{top(S_i)}, s)$  and
  - add the edge  $(u_{top(S_i)}, u_{top(S_{i-1})})$  in  $E(F_2[\pi^*])$ ;
3. Return the graph  $F_2[\pi^*]$ ;

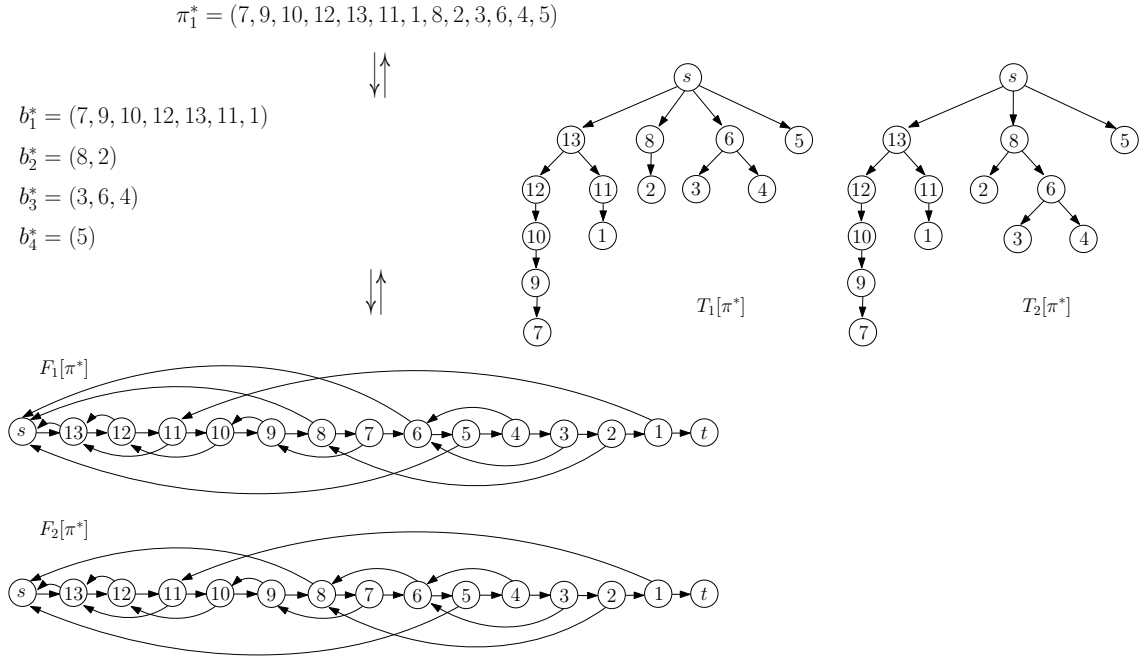


Figure 2: The main structures used or constructed by the codec algorithms `Encode_SiP.to.RPG-Bitonic-2` and `Decode_RPG.to.SiP-Bitonic-2`.

We next describe the corresponding decoding algorithm for extracting the permutation  $\pi^*$  from the flow-graph  $F_2[\pi^*]$ .

**Algorithm** `Decode_RPG.to.SiP-Bitonic-2`

1. Execute Steps 1 and 2 of algorithm `Decode_RPG.to.SiP-Bitonic-1` on graph  $F_2[\pi^*]$  and compute the directed graph  $T_2[\pi^*]$  and the node set  $R$ ;
2. Compute the node set  
 $R' = \{u_j \mid (u_i, u_j) \in T_2[\pi^*] \text{ and } \text{outdeg}(u_j) \geq 2\},$   
delete the directed edges  $(u_i, u_j)$  from the graph  $T_2[\pi^*]$ , and set  $R \leftarrow R \cup R'$ ;
3. Execute Steps 3, 4 and 5 of the decode algorithm `Decode_RPG.to.SiP-Bitonic-1` and compute the sequences  $b_1^*, b_2^*, \dots, b_k^*$ ;
4. Return  $\pi^* = (b_1^*)^R || b_2^* || \dots || b_{k-1}^* || (b_k^*)^R$ ;

In the example of Figure 2,  $R = \{13, 8, 5\}$  and  $R' = \{6\}$ . Recall that, the self-inverting permutation which encodes watermark  $w$  is of length  $n^* = 2n + 1$ , where  $n$  is the binary length of the watermark number  $w$ , while the reducible permutation flow-graph  $F_2[\pi^*]$  is of size  $n^* + 2$ .

The results of this section concerning the correctness and the time and space complexity of both algorithms are summarized in the following theorem.

**Theorem 4.1** *The algorithm `Encode_SiP.to.RPG-Bitonic-2` encodes a permutation  $\pi^*$  into a reducible permutation flow-graph  $F_2[\pi^*]$  in  $O(n)$  time and space and the corresponding decoding*



algorithm `Decode_RPG.to.SIP-Bitonic-2` correctly extract  $\pi^*$  from the flow-graph  $F_2[\pi^*]$  within the same time and space complexity.

## 5 Concluding Remarks

In the last decade, a wide range of software watermarking techniques has been proposed among which the graph-based methods that encode watermark numbers as graphs whose structure resembles that of real program graphs. Recently, Chroni and Nikolopoulos [2, 4] proposed several algorithms for multiple encoding a watermark into a graph-structure: an integer (i.e., a watermark) is encoded first into a self-inverting permutation  $\pi^*$  and then into several reducible permutation graphs using Cographs [2] and Heap-ordered trees [4].

Following up on our recently proposed methods, in this paper, we extended the class of graph-structures by proposing two different reducible permutation flow-graphs  $F_1[\pi^*]$  and  $F_2[\pi^*]$  incorporating important structural properties which are derived from the bitonic subsequences forming the self-inverting permutation  $\pi^*$ . These new flow-graphs enrich the repository of graphs available for multiple encoding a watermark number and, thus, it increases our ability to select a graph structure more similar to the structure of a given application program  $P$  thereby enhancing the resilience of our codec system to attacks.

An interesting open question is whether the properties of the bitonic subsequences forming the self-inverting permutation  $\pi^*$  can help develop efficient graph structures having more similar structure to that of a given application program  $P$ . Can we use some of the bitonic subsequences  $b_1^*, b_2^*, \dots, b_k^*$  of permutation  $\pi^*$  or part of them in order to efficiently encode and decode a self-inverting permutation  $\pi^*$  into a reducible permutation flow-graph  $F[\pi^*]$ ? we leave it as an open problem for future investigation.

## References

- [1] M. Chroni and S.D. Nikolopoulos. Encoding watermark integers as self-inverting permutations. International Conference on Computer Systems and Technologies (CompSysTech'10), ACM ICPS 471, 125–130, 2010
- [2] M. Chroni and S.D. Nikolopoulos. Encoding watermark numbers as cographs using self-inverting permutations. International Conference on Computer Systems and Technologies (CompSysTech'11), ACM ICPS 578, 142–148, 2011
- [3] M. Chroni and S.D. Nikolopoulos. An efficient graph codec system for software watermarking. Proc. 36th IEEE Conference on Computers, Software, and Applications (COMPSAC'12), IEEE Proceedings, 595–600, 2012
- [4] M. Chroni and S.D. Nikolopoulos. Encoding numbers into reducible permutation graphs using heap-ordered trees. Proc. 19th Panhellenic Conference on Informatics, ACM, 2015
- [5] C. Collberg and J. Nagra. Surreptitious Software. Addison-Wesley, 2010
- [6] C. Collberg and C. Thomborson. Software watermarking: models and dynamic embeddings. Proc. 26th ACM SIGPLAN-SIGACT on Principles of Programming Languages (POPL'99), 311–324, 1999
- [7] C. Collberg, C. Thomborson, and D. Low. On the limits of software watermarking. Department of Computer Science, The University of Auckland, Technical Report No 164, 1998

- [8] R.L. Davidson and N. Myhrvold. Method and system for generating and auditing a signature for a computer program. US Patent 5.559.884, Microsoft Corporation, 1996
- [9] D. Eppstein, M. T. Goodrich, J. Lam, N. Mamano, M. Mitzenmacher and M. Torres. Models and Algorithms for Graph Watermarking. arXiv preprint arXiv:1605.09425, 2016
- [10] R. Ghiya and L.J. Hendren. Is it a tree, a DAG, or a cyclic graph? a shape analysis for heapdirected pointers in c. Proc. 23rd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL'96), 1–15, 1996
- [11] D. Grover. The Protection of Computer Software - Its Technology and Applications. Cambridge University Press, New York, 1997
- [12] M.S. Hecht and J.D. Ullman. Flow graph reducibilit. SIAM J. Computing 1, 188–202, 1972
- [13] M.S. Hecht and J.D. Ullman. Characterizations of reducible flow graphs. Journal of the ACM 21, 367–375, 1974
- [14] S.A. Moskowitz and M. Cooperman. Method for stegacipher protection of computer code. US Patent 5.745.569, 1996
- [15] G. Myles and C. Collberg. Software watermarking via opaque predicates: Implementation, analysis, and attacks. Electronic Commerce Research 6, 155–171, 2006
- [16] A. S. Panah, R. van Schyndel, T. Sellis and E. Bertino. On the properties of non-media digital watermarking: A review of state of the art techniques. DOI 10.1109/ACCESS.2016.2570812. IEEE, 2016
- [17] P. Samson. Apparatus and method for serializing and validating copies of computer software. US Patent 5.287.408, 1994
- [18] R. Sedgewick and P. Flajolet. An Introduction to the Analysis of Algorithms. Addison-Wesley, 1996
- [19] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto. Design and evaluation of birthmarks for detecting theft of Java programs. Proc. Int'l Conference on Software Engineering (IASTED SE'04), 569–575, 2004
- [20] L. Zhang, Y. Yang, X. Niu, and S. Niu. A survey on software watermarking. Journal of Software 14, 268–277, 2003
- [21] W. Zhu, C. Thomborson, and F.Y. Wang. A survey of software watermarking. Proc. IEEE Int'l Conference on Intelligence and Security Informatics (ISI'05), LNCS 3495, 454–458, 2005